FAA National Software Conference, June 2001
Object-Oriented Technology and Certification

Overview of OOT and certification concerns

Salah Obeid
I-Logix
Sobeid@ilogix.com
480-460-9001

I-Logix

Presentation Outline

- Overview of Object-Oriented Technology
- Overview of design notation UML
- Overview of auto code generation
- Certification Concerns of OOT
- Summary

Unified Modeling Language (UML)

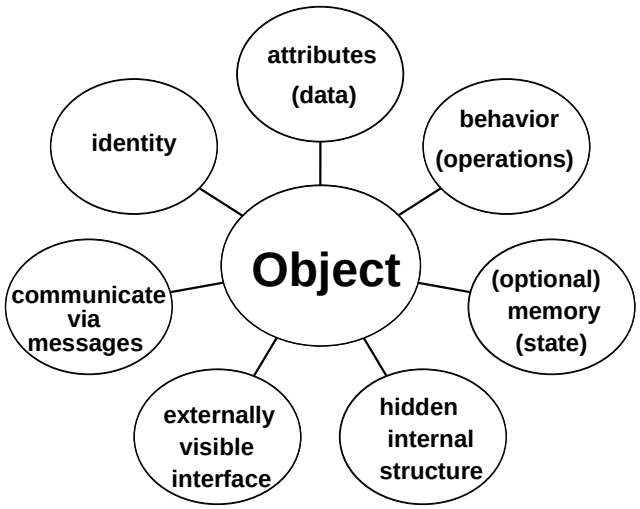
- Provide system architects working on object analysis and design with consistent language for:
  - Specifying
  - Visualizing
  - Constructing
  - Documenting

  The artifacts of software systems

What Is an *Object* ?

- Several definitions are available:
  - An object is a real-world or conceptual thing that has autonomy
  - An object is a cohesive entity consisting of data and the operations that act on those data
  - An object is a thing that has an interface that enforces protection of the encapsulation of its internal structure

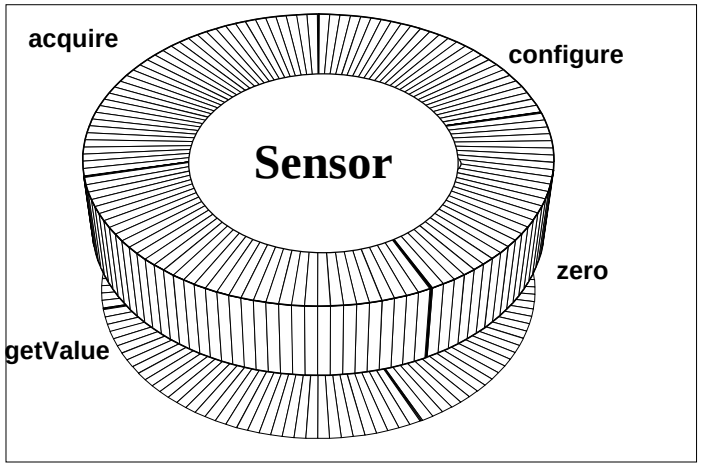
What Is an *Object* ?



Object Interface

- Objects present a public interface to the world
- Internal details are hidden from view and access is controlled
- Use of interfaces forces adherence to an object's protocol, consisting of
  - ✓ Pre-conditional invariants
  - ✓ Post-conditional invariants
  - ✓ Signature

Object Interface

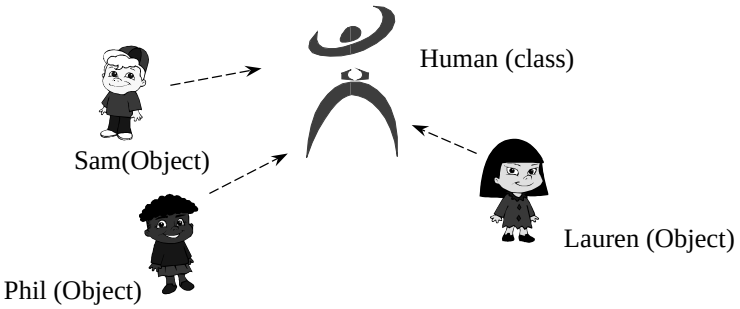


Object Behavior

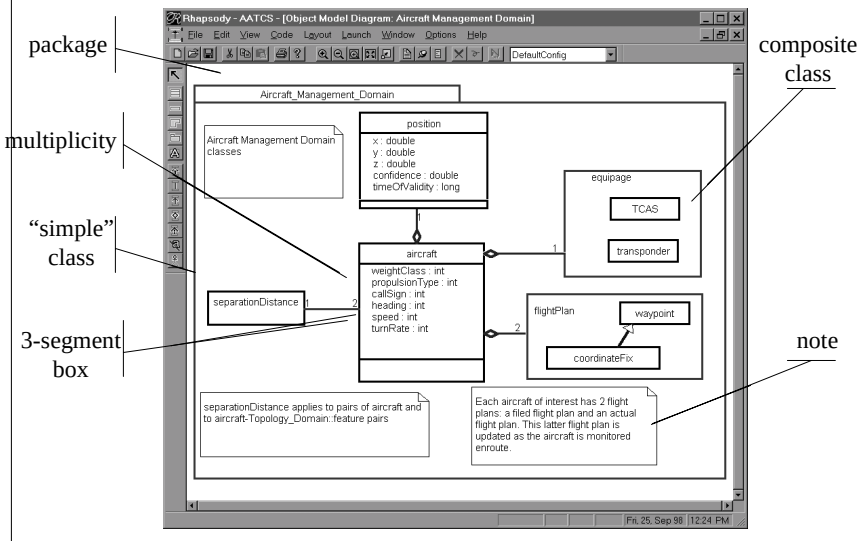
- All objects must know how to create and destroy themselves
- Only the appropriate semantics should be exported
- Do not reveal internal structure through the operations
- Object behaviors are known as *operations*
- A *method* is the implementation of an operation within a class

Classes

- Class is the ***definition*** or ***specification*** of an object
- An object is an ***instance*** of a class
- An object has the attributes and behaviors defined by its class
- It is common to have many instances of a class at the same time



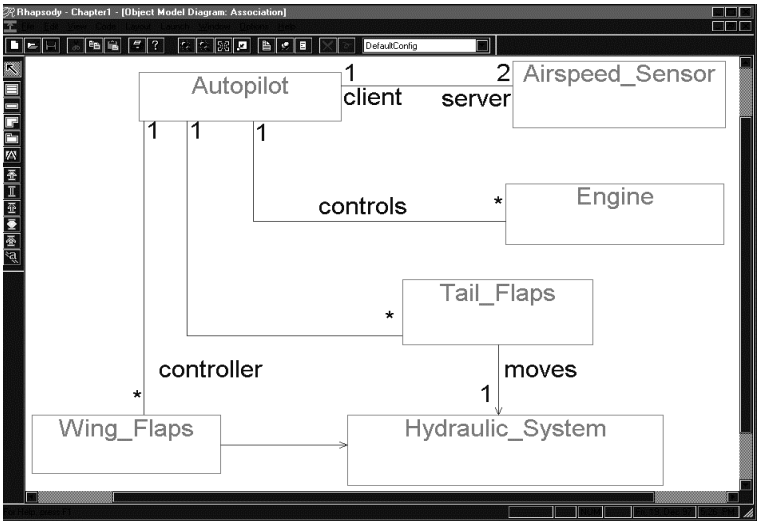
Class Diagrams



Relationships

- Relationships allow objects to communicate at run-time
- Objects may use the facilities of other objects with an ***association***
- Objects may contain other objects with an ***aggregation***
- Objects may *strongly* aggregate others via ***composition***
- Classes may derive attributes and behaviors from other classes with a ***generalization***

Classes with Associations



Types of Behavior

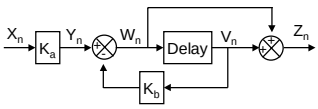
- Behavior can be *simple*
  simple behavior does not depend on the object's history
- Behavior can be *continuous*
  continuous behavior depends on the object's history but in a smooth, continuous fashion
- Behavior can be *state-driven*
  state-driven behavior means that the object's behavior can be divided into disjoint sets

Simple Behavior

- Simple behavior is not affected by the object's history
  - cos(x)
  - getTemperature( )
  - setVoltage(v)
  - Max(a,b)
  - $\int_a^b e^{-x^2} \, dx$
- May be represented by activity diagrams, if desired

Continuous Behavior

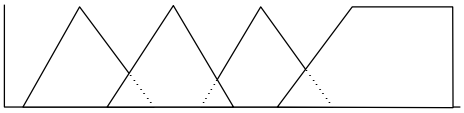
- Object's behavior depends on history in a continuous way
  - PID control loop



  - Digital filter

$$f_j = \frac{d_j + d_{j-1} + d_{j-2} + d_{j-3}}{4}$$

  - Fuzzy logic
    uses partial set membership to compute a smooth, continuous output



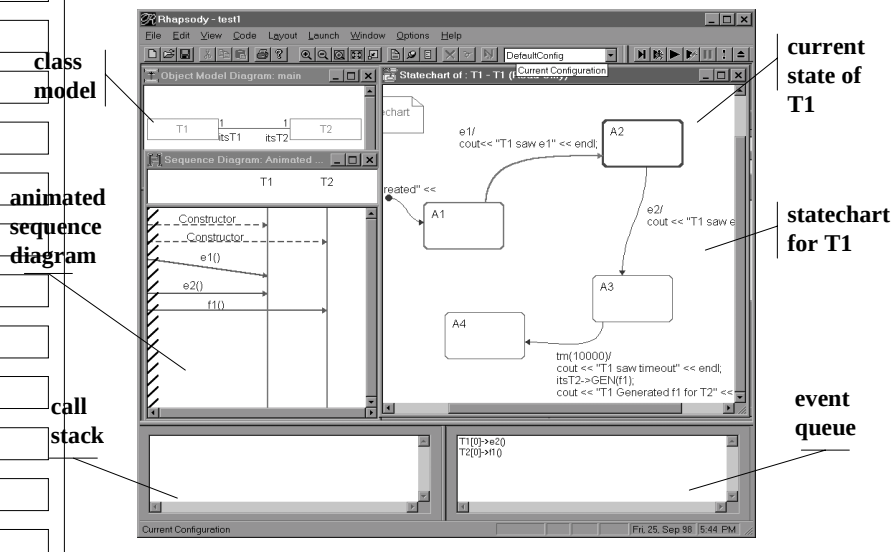
State Behavior

- Object exhibits discontinuous modes of behavior
  - **In "therapy mode" delivery anesthetic agent based on knob position**
  - **In "service mode" select service function based on knob position**
  - **In "startup mode" ignore the knob turns**

State Machines Simplify Your Life

- A state machine is an abstraction of the desired behavior of a system with actions, activities, and constraints
  - An object's behavioral possibilities are defined in a primitive way by its defined operations
  - A state machine constrains the use of those operations into particular sequences
  - Thus constrained, the object's behavior is more understandable and more easily tested
- State machines are a more abstract view of behavior, based in the object's perspective not the implementation!

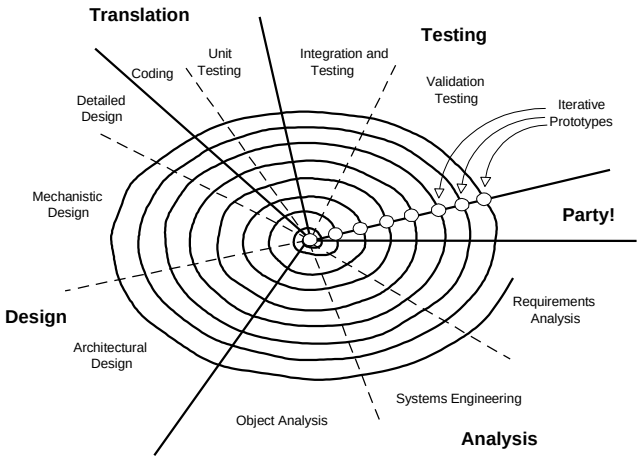
State Machines Are Executable

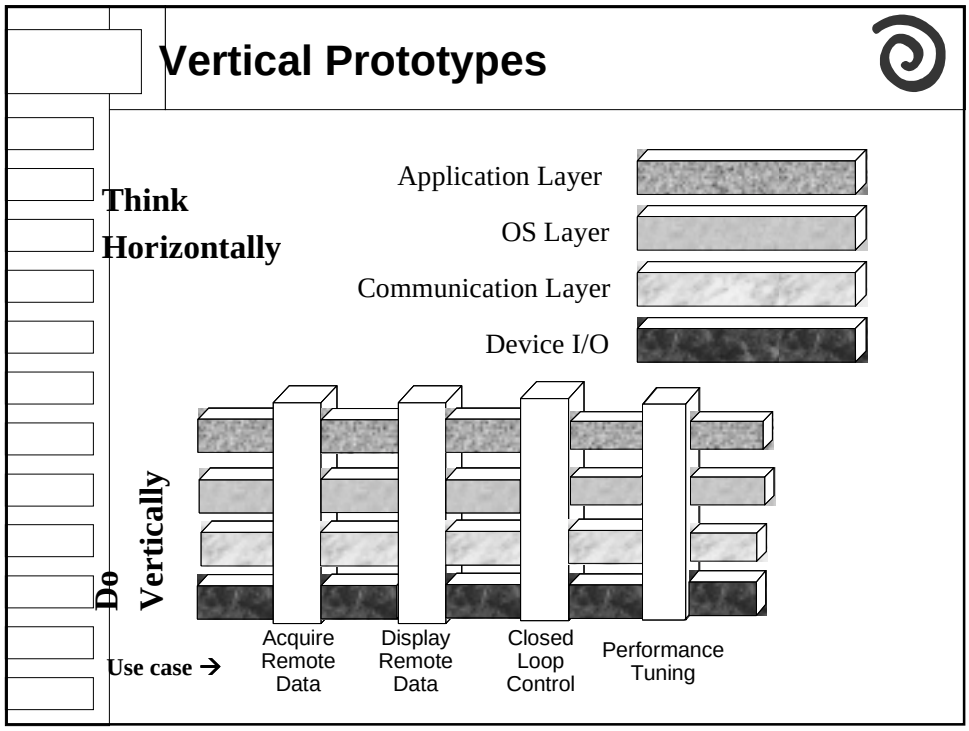


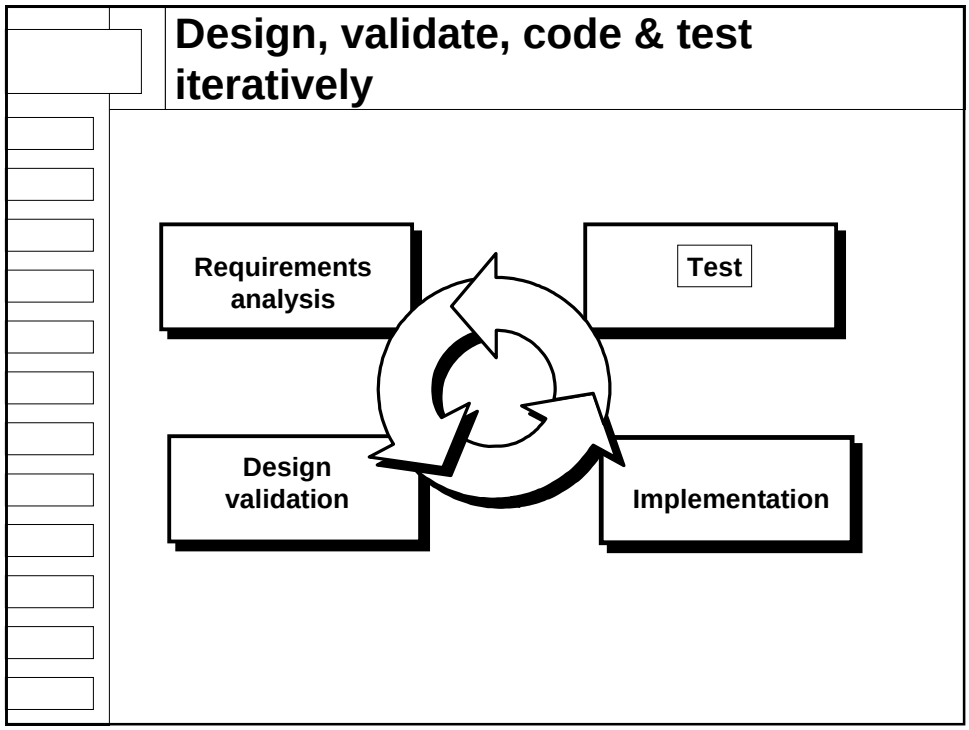
Easy Testing

- Unit testing is fundamentally at the level of the object (or function)
- *Finite state machines* (FSMs) can be subdivided into states for testing
  - Each state has a smaller set of input and output conditions
  - The overall testing is decreased because of low coupling among states

Iterative process




Vertical Prototypes
Think Horizontally
Application Layer
OS Layer
Communication Layer
Device I/O
Do Vertically
Use case →
Acquire Remote Data
Display Remote Data
Closed Loop Control
Performance Tuning


Design, validate, code & test iteratively
Requirements analysis
Test
Design validation
Implementation

Auto Code Generation

- UML is defined rigorously to permit automatic generation of code from object models
  - Class structures
    - Classes
    - Relations
  - Full class behavioral models
    - State Charts
    - Sequence diagrams

Auto code generation

- The goal is to generate a framework that can be verified according to the DO-178B guidelines:
  - Develop a design guideline
  - For each design artifact determine the code generation and testing strategy

Some Certification Concerns

Inheritance

- Single inheritance
  - Simple
  - The test case will have to be run on the inherited class
- Multiple inheritance
  - More complicated
  - Problematic
  - Should be avoided

Overload

- To overload a function
  - Provide another function with the same name in the same scope but with different parameter types

Override

- To override a virtual function
  - Provide another function with the same name and the same parameter type in a derived class
- Never change the default parameters of overridden inherited functions

Inline

- A tradeoff between code size and performance
- Use it wisely
- Should be tested wherever it's used

Dead Code

- No new news
  - Should be removed or justified

Deactivated Code

- Re-use issues
  - Components level re-use
  - Class level re-use
- Test cases should be added for complete testing and complete structure coverage

Summary

- Working at the design level
  - Use of visual modeling
  - Working at the correct abstract level
  - Using an iterative process
- Certification issues with OOT
- On going activities